The design of a Disk Resource Manager
(and its relationship to GSDAs and File Replication)

Arie Shoshani, LBNL

October, 2000

A Disk Resource Manager (DRM) is a component that controls the use of a shared disk cache in a data grid. As a shared disk, it can be used by multiple clients and therefore a DRM usually has some policy that determines how much of the disk cache can be used by a given client, how long to keep a file in disk once it was cached, and which files to remove when space is needed.

A typical scenario

To illustrate the functionality of a DRM, consider the typical scenario where a client wants a file to be moved from a source DRM (sDRM) to a target DRM (tDRM). By a client we mean an application program, or another agent, such as a "request manager", acting on behalf of the user. Before we consider the question of allocating space, we need to address an important question: which component will initiate the file transfer. There are 3 possibilities:
a) The tDRM "pulls" the file (e.g. by doing an FTP-get").
b) The sDRM "pushes" the file (e.g. by doing an FTP-put).
c) The client initiates a 3rd party transfer from sDRM to tDRM (assuming this capability is available).

The considerations of "pull vs. push" were already discussed quite well by Andy Hanushevsky in the document "Data Management in Data Grids". We came to the same conclusion of having a "pull" model; we had similar reasons for that. Assuming that a DRM manages a disk cache under some local site control, that site will have its own security and policy requirements. Thus, it makes sense that the site manager will not want to give permission to external agents to write into its space. By using the pull model, the DRM is in full control of its space.

Now, assuming a pull model, the scenario is then changed to the client making a request to the tDRM to get a file from the sDRM. We assume here that the client (such as a "request manager") already checked with the replica catalog as to where to get the file from, and it has a source URL (sURL) for it. The client also provides the target URL (tURL) – the location where the file should be moved to. The following actions will take place:

a) The client asked the tDRM to get a file from sURL to tURL.
b) tDRM tries to allocate space for the file. If no space is available, or it is otherwise busy, it can choose to queue the request for later execution.
c) If space is found, the tDRM contacts the sDRM with a request to pin this file.

d) If sDRM has the file (the file could have been removed in the meantime), it returns OK status. Otherwise, it returns fail.
e) Assuming the file was pinned by sDRM, then tDRM pulls the file (e.g. with a grid-ftp GET). When completed, the DRM checks that the file moved properly.
f) tDRM notifies the client that the file is available. If the request was a non-blocking call, tDRM calls the client. Alternatively, the client can sent tDRM a status request to find when the file arrives.
d) tDRM notifies sDRM that the file can be "released". It is the choice of sDRM whether to keep the file in the cache, or actually remove it. Typically, files are not removed till space is needed.

Discussion of various functions

There are several issues that have to be dealt with when designing the DRM functionality. These include getting a file into the disk from another source, pinning a file, storing a file by a client, how to enforce time outs, queuing requests by the DRM, registration of files in the replica catalog, space reservations, status of a file transfer request, and collecting and providing statistics on the cache availability and usage. We describe each below. These functions form the basis for a DRM IDL provided in the Appendix.

1. Getting a file into the disk from another source
To recap the scenario above, a target DRM must be able to support a request for *getting* a file from another disk into its disk cache. It is assumed that the location of the file is known to the client because it is registered with the replica catalog. Normally, a file will be pulled from another (source) DRM, but in principle files can be pulled from any source as long as it supports some file transfer protocol. A request to the DRM to get this file requires that the DRM makes space for the file in the disk cache, if necessary by removing other unused files, requesting that the file at the source DRM be pinned (if such a capability exists), initiate a file transfer from the source to its disk, monitoring the progress of the transfer, and notifying the caller when the file has been fully transferred. After a file has been transferred, the DRM may register the file with the file catalog, so that other clients can use it. A requested file may be found in the DRM's disk cache because another user has previously requested it. In this case, it is wise to share the file. We will discuss later the method for sharing files without copying them into each client's space.

2. Pinning a file
Assuming that a (source) DRM can hold files temporarily, it is necessary to ask it to pin a file before a file transfer from it is initiated. This is part of facilitating file sharing between clients. Of course, one can skip this step, but then we take a chance that the file may be removed in mid transfer, or may not be there when the transfer is initiated. This is true, even if the source is well behaved in that it removes its entry from the replica catalog, because of an obvious race condition: the client may have gotten the replica reference and did not start the transfer by the time it is removed. If a source system has no DRM, one can assume that the file will be there for a sufficiently long time and it is

safe to perform a file transfer without pinning. Usually, a file being pinned is expected to be found in the cache, but there is no guarantee of that. Furthermore, it is up to the local DRM's policy as to how long it keeps a file pinned. That may be governed by a time out policy, discussed next.

3. Time-outs

A well behaved client is expected to release a file after it is done using it. However, a DRM cannot rely on that, and must have a policy for removing files. It is up to the local administrator of each DRM to determine the policy for a time-out of a file. There may be a different time-out policy for files transferred into the DRM for the client's use from the time-out policy for file pinned by the DRM. A time-out policy is a best-effort promise that a file will be in cache for the time out period, but it is not a guarantee. Therefore DRMs need to be designed to handle such mishaps. Beyond the time-out period, the file may or may not be found in cache, depending on its sharing by other clients, and the need to release space. The time out enforcement is therefore per_file_per_user. The time-out left can be reported to the client when the file is cached or pinned, or later in response to a status, but it must be understood that it is not a guarantee.

4. Storing a file by a client

Another function of a DRM is to store files that clients generate. Typically, such files are stored temporarily and then transferred to other sites, perhaps a tape system. Thus, there is no need to register such files in the replica catalog. This function is similar to getting a file as far as making space for the file, but the DRM does not have to pin a file a the source. The client, who is obviously the owner of the file, is responsible to register it with the file catalog, if he/she chooses to do that.

5. Queuing requests

In a busy system, it is possible that a file transfer request cannot be accomplished at the time it is requested, most likely because space is not found immediately, or because a limit on the number of concurrent transfer that the DRM can perform. In this case, if an error "system busy" is returned to the client, the client will try repeatedly until the request is accepted or it may give up. To avoid this situation, a DRM can help by queuing requests. Rather than return a "system busy" response, it can queue the request, and return a status with an estimated time before the file will be scheduled for trnasfer. The client can then accept the estimated time or abort the request. This feature is not essential for a DRM, but having it can cut down on repeated requests and can be used to enforce policy and fairness of service of clients' requests.

5. Registration of files in the replica catalog

If the DRM is managing a shared disk cache, then any file stored in that system can potentially be needed by other clients. Thus, after a file is transferred to the DRM's disk, it should be registered in the Replica Catalog, so that other clients know about its existence. The choice to register a file is part of the DRM's policy. Also, the frequency of registration is a local policy. The frequency of registration can be either immediate after a file was transferred, or one can have a policy to update the replica catalog periodically. Similarly, before a shared file is removed by DRM (to make space for new files) the DRM should remove the replica entry from the file catalog. In this case, it is wise to remove the replica entry before the file is physically removed, but as mentioned in point 2 above, this does not guarantee that the file will not be requested after is removed.

6. Space reservations

Space reservations can be a problematic function to provide. On the one hand, it makes sense for clients to make plans to use a certain amount of space at a certain time for a certain duration. On the other hand, such reservations can be very wasteful of space, since a DRM must keep the space unused to guarantee a reservation. It seems that for reservations to be effective, some kind of a cost must be charged to the clients, so they have an incentive to act responsibly. A responsible behavior means that the clients use the space assigned as soon as it is assigned, and release it as soon as they do not need the space. Another aspect of space assignment is how to advertise or describe the available space, since it is a function of time. This information is needed for request planning. At a minimum every DRM supporting reservations should be able to respond to a request for space reservation for a certain time and duration either positively or suggest the earliest time that such a reservation can be made.

7. Status of a file transfer request

After a request for file caching is made, the DRM should be capable of providing the status of the request. This includes information on whether the request is still queued, whether the transfer is in progress, and how much of the file was transferred so far. In case that the request is still queued, a time estimate should be provided as to when the request will be executed.

8. Providing statistics on the cache availability and usage

This functionality is necessary in order for clients of the DRM to plan how to use the DRM. This includes statistics on past usage of the cache and the size of its request queue over time. Such statistics could be used to determine if the cache is overused and causes a bottleneck in the data grid.

9. Sharing a file in the DRM's cache

When a request for a file is made to a DRM, it should check first if the file is already in the cache, and can be shared (assuming the file is "read-only). For this reason, the DRM should keep track of the logical file name of files it keeps. Similarly, a logical file name should also be provided by the client when it request a "get file".

How does a DRM facilitate sharing of a file? Suppose we have 2 clients interested in a file. If the file is stored in the tURL provided by the first client, then another copy of the file must be made into the tURL provided by the second client. If we put he file in a shared area, the first client could delete the file while the second client is accessing it. Thus, a shared file should be under the control of the DRM, and stored in a disk partition that belongs to the DRM. Our planned solution is to link (i.e. a unix link) the tURL to the file stored in the DRM's cache. This solution also permit advertising the existence of the file in the replica catalog.

10. Coordinating on the transfer protocol
In discussions with people from Fermi (mostly with Rich Wellner) on the HRM (Hierarchical Resource Manager – that manages file staging from tape to a shared disk) interface and implementation, it was suggested that the coordination of a transfer protocol is made as part of the interaction between the client and the HRM. This was eventually made part of the HRM IDL. The idea is that the client provides a list of protocols in order of preference that it can support (e.g. HTTP, GRID-FTP, NCFTP, …), and the HRM respond with the first protocol from that list it can support. We think that the same concept could be useful in the communication between DRMs. This leaves the option for local sites to set up their preferred protocols and security requirements.

11. Security concerns
Since tDRMs and sDRMs communicate with each other, and perform requests from the grid services to transfer files, it is easiest if the DRMs are authenticated as special trusted agents. This will prevent the need for each user to be authenticated in each DRM site. However, this puts the burden of authenticating a user and checking his/her permission to get a file on some intermediary agent.

The coordination between the DRMs is only control information (such as "pin a file" "release a file", "status of a file", etc.) It is not clear to me whether this information should be treated as secure. On the other hand, file transfers should definitely be done securely, including when they are moved by trusted agents.

The relationship to GSDAs

In our discussions with Andy, we found that the concepts of a DRM are well suited to GSDAs (Grid Storage Domain Agent). GSDAs are the same concept as the "known grid node" used in Doug Olson's white paper on file replication. The main additional function that a GSDA performs in addition to a DRM is the "redirection" function. The use of a GSDA permits the registration of a file in the replica catalog as a "durable URL" (Doug's terminology) or "storage domain locations" (Andy's terminology). This understanding will permit GSDAs to use the same interface as a DRM, except that the client need to be prepared to handle "redirection".

The relationship to File Replication Service

Suppose there is a need to replicate a set of files. This may be a periodic need of bulk replication (as Doug suggested) or dynamic need as files move around because of clients demand. I view this as similar needs, except that we might want to introduce the concept of a "permanent replica" – i.e. a replica under the control of the administrator only. Dynamic replicas are under the control of the DRM (or HRM) that stores them.

As Doug suggested, a "File Replication Service" may perform several functions in addition to actually request to transfer files, such as check that the file does already exist in the targeted site, and register file in the replica catalog after the file is transferred.

However, the actual file transfer can be performed with DRMs and HRMs. The "Stage a file to a local grid node" is a request to an HRM. Once the file has been staged, the file transfer request could be coordinated with DRMs. The DRMs allocate the space and pin files as described above, and use the grid transfer service (such as grid-ftp) to pull the files. Again, if GSDAs are available, then the communication is with a GSDA that redirects the transfer request to a DRM.

I believe that PPDG will benefit from a common interface definition for DRMs and GSDAs. Each local system can be as simple as a thin layer on top of an FTP server, or a sophisticated as managing multiple DRMs and HRMs. The experience with using a single HRM interface to access radically different systems (HPSS at LBNL, and SAM-Enstore at Fermi) illustrates this point.

Acknowledgement

This document is a result of many discussions I had with Alex Sim and Andreas Mueller at LBNL, as well as a day long meeting with Miron Livny, and another meeting with Andy Hanushevsky.

Appendix: a proposed DRM IDL (draft)

```
// Andreas Mueller Amueller@lbl.gov
// Arie Shoshani shoshani@lbl.gov
// Alex Sim Asim@lbl.gov
// Lawrence Berkeley National Laboratory
// Octoberber 2000
// Purpose: defines interface between Disk Resource Manager (server) and
clients

#ifndef DRM_IDL
#define DRM_IDL

//#include <ppdgDefs.idl>

struct timer {
	short hour;
	short minute;
	short second;
};

struct DRMFileInfo {
	string logFileName;
	string UID;
};
```

```
enum drmStatus {
        DRM_FILE_EXISTS,
        DRM_TRANSFER_IN_PROCESS,
        DRM_FILE_TRANSFER_DONE,
        DRM_FILE_TRANSFER_FAILED,
        DRM_NOT_ENOUGH_SPACE,
        DRM_FILE_NOT_IN_SOURCE,
        DRM_RELEASE_DONE,
        DRM_NOT_REQUESTER,
        DRM_RELEASE_FAILED,
        DRM_PURGE_DONE,
        DRM_PURGE_FAILED,
        DRM_DISK_DOWN,
        DRM_DISK_ERROR,
        DRM_T_DISK_DOWN,
        DRM_T_DISK_ERROR,
        DRM_REQUEST_QUEUED,
        DRM_REQUEST_FAILED,
        DRM_FILE_TIMED_OUT
};

enum drmPINStatus {
        PINNED,
        PINNING_FOR_USER_REFUSED,
        PINNING_FAILED,
        PIN_RELEASE_FAILED,
        ALREADY_PINNED_BY_USER,
        FILESIZE_MISMATCHES
};

interface DiskResourceManager {
        void hello();
        boolean areYouAlive();

        boolean getFile(in DRMFileInfo fileUID,in string S_URL,
                              in string T_URL, in double fileSize);

        boolean getFileNonBlock(in DRMFileInfo fileUID, in string S_URL,
                in string T_URL,in double fileSize, out string client_ref);
        boolean abortTransfer(in DRMFileInfo fileUID, in short
                         ftpProcessnumber, in string T_URL);

        void getStatus(in DRMFileInfo fileUID, in string T_URL,
              out double transfered_bytes, out double remain_bytes);
```

```
    boolean releaseFile(in string S_URL,in string UID);
    boolean pinFile(in string S_URL, in string UID, out short PIN);
    double getPIN(in string S_URL, in string UID);
};

#endif // DRM_IDL
```